Bahria University,

Karachi Campus

## LAB EXPERIMENT NO.

9

## LIST OF TASKS

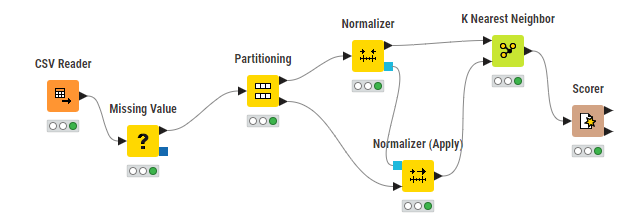
|  |  |
| --- | --- |
| **TASK NO** | **OBJECTIVE** |
| **1** | Perform KNIME on IRIS dataset |
| **2** | Implement Knime |
| **3** | Implement a Movie Recommendation System Using K-Nearest Neighbors (KNN) |

Submitted On:

24 April 2024

(Date: DD/MM/YY)

**TASK#1:** Implement KNN for Iris Flower Classification in KNIME



Load the Iris dataset into KNIME



Preprocess the data as necessary, including handling missing values, if any.

A screenshot of a computer

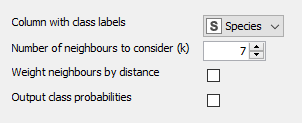
Description automatically generated

Split the dataset into training and testing sets using the Split Data node.

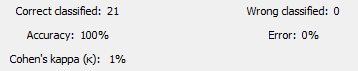
A white background with black and white clouds

Description automatically generated with medium confidence

Configure the KNN Learner node to train the KNN classifier with a chosen number of neighbors and distance metric. 5. Use the trained KNN model to make predictions on the testing set



Evaluate the performance of the classifier using the Scorer node to calculate accuracy.

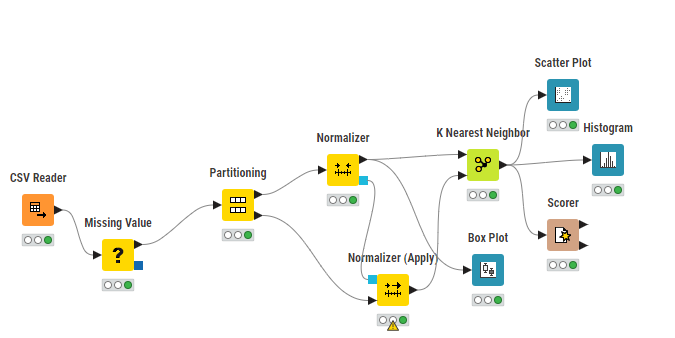


Optionally, visualize the results using appropriate nodes such as the Interactive Table or Confusion Matrix.

A screenshot of a computer

Description automatically generated

**TASK # 2: :** Implement KNN



Load the Iris dataset into KNIME



Preprocess the data as necessary, including handling missing values, if any.

A screenshot of a computer

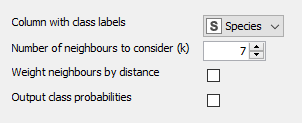
Description automatically generated

Split the dataset into training and testing sets using the Split Data node.

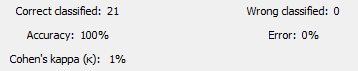
A white background with black and white clouds

Description automatically generated with medium confidence

Configure the KNN Learner node to train the KNN classifier with a chosen number of neighbors and distance metric. 5. Use the trained KNN model to make predictions on the testing set



Evaluate the performance of the classifier using the Scorer node to calculate accuracy.



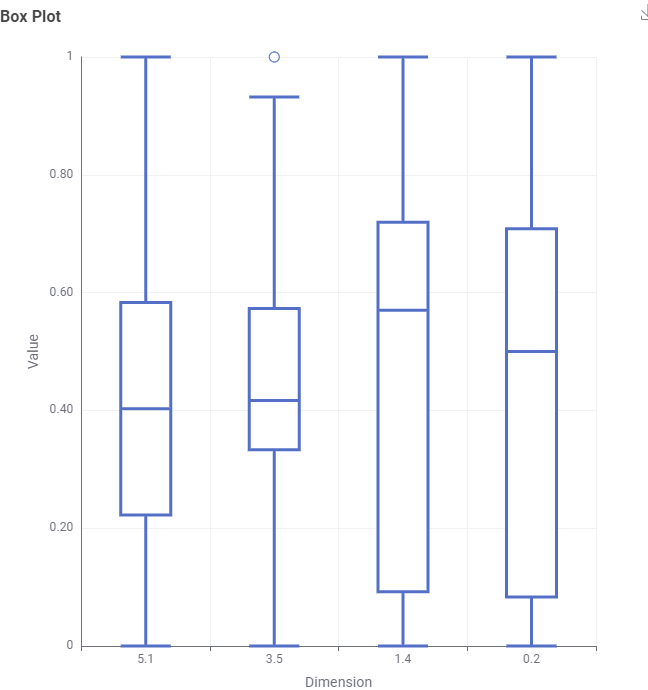
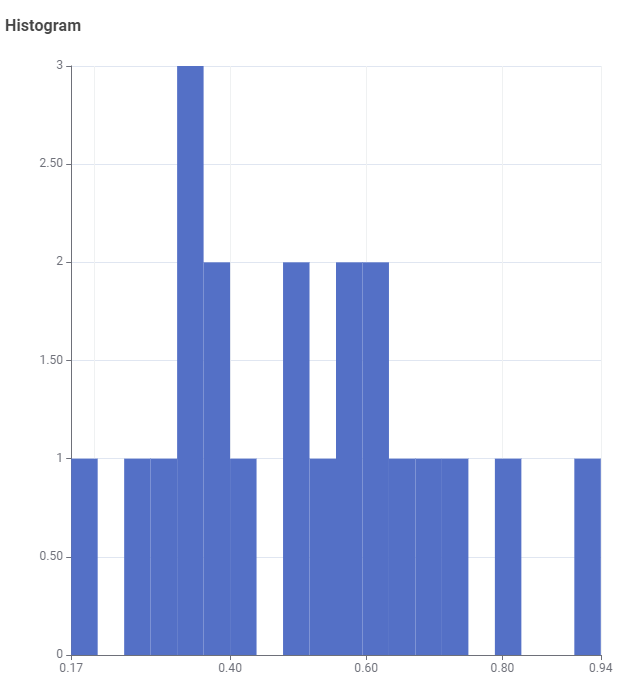
Optionally, visualize the results using appropriate nodes such as the Interactive Table or Confusion Matrix.

A screenshot of a computer

Description automatically generated

Visualize the results using appropriate plots or visualizations.

A screen shot of a graph

Description automatically generated

TASK # 3: Implement a Movie Recommendation System Using K-Nearest Neighbors (KNN)

import pandas as pd

import matplotlib.pyplot as plt

plt.style.use('fivethirtyeight')

import seaborn as sns

import numpy as np

import json

import warnings

warnings.filterwarnings('ignore')

from scipy import spatial

movies = pd.read\_csv('../input/tmdb-movie-metadata/tmdb\_5000\_movies.csv')

credits = pd.read\_csv('../input/tmdb-movie-metadata/tmdb\_5000\_credits.csv')

movies['genres'] = movies['genres'].apply(json.loads)

for index, i in zip(movies.index, movies['genres']):

    list1 = []

    for j in range(len(i)):

        list1.append(i[j]['name'])  # the key 'name' contains the name of the genre

    movies.loc[index, 'genres'] = str(list1)

movies['keywords'] = movies['keywords'].apply(json.loads)

for index, i in zip(movies.index, movies['keywords']):

    list1 = []

    for j in range(len(i)):

        list1.append(i[j]['name'])

    movies.loc[index, 'keywords'] = str(list1)

movies['production\_companies'] = movies['production\_companies'].apply(json.loads)

for index, i in zip(movies.index, movies['production\_companies']):

    list1 = []

    for j in range(len(i)):

        list1.append(i[j]['name'])

    movies.loc[index, 'production\_companies'] = str(list1)

credits['cast'] = credits['cast'].apply(json.loads)

for index, i in zip(credits.index, credits['cast']):

    list1 = []

    for j in range(len(i)):

        list1.append(i[j]['name'])

    credits.loc[index, 'cast'] = str(list1)

credits['crew'] = credits['crew'].apply(json.loads)

def director(x):

    for i in x:

        if i['job'] == 'Director':

            return i['name']

credits['crew'] = credits['crew'].apply(director)

credits.rename(columns={'crew': 'director'}, inplace=True)

movies = movies.merge(credits, left\_on='id', right\_on='movie\_id', how='left')

movies = movies[['id', 'original\_title', 'genres', 'cast', 'vote\_average', 'director', 'keywords']]

movies['genres'] = movies['genres'].str.strip('[]').str.replace(' ', '').str.replace("'", '')

movies['genres'] = movies['genres'].str.split(',')

plt.subplots(figsize=(12,10))

list1 = []

for i in movies['genres']:

    list1.extend(i)

ax = pd.Series(list1).value\_counts()[:10].sort\_values(ascending=True).plot.barh(width=0.9, color=sns.color\_palette('hls', 10))

for i, v in enumerate(pd.Series(list1).value\_counts()[:10].sort\_values(ascending=True).values):

    ax.text(.8, i, v, fontsize=12, color='white', weight='bold')

plt.title('Top Genres')

plt.show()

for i, j in zip(movies['genres'], movies.index):

    list2 = []

    list2 = i

    list2.sort()

    movies.loc[j, 'genres'] = str(list2)

movies['genres'] = movies['genres'].str.strip('[]').str.replace(' ', '').str.replace("'", '')

movies['genres'] = movies['genres'].str.split(',')

genreList = []

for index, row in movies.iterrows():

    genres = row["genres"]

    for genre in genres:

        if genre not in genreList:

            genreList.append(genre)

def binary(genre\_list):

    binaryList = []

    for genre in genreList:

        if genre in genre\_list:

            binaryList.append(1)

        else:

            binaryList.append(0)

    return binaryList

movies['genres\_bin'] = movies['genres'].apply(lambda x: binary(x))

movies['cast'] = movies['cast'].str.strip('[]').str.replace(' ', '').str.replace("'", '').str.replace('"', '')

movies['cast'] = movies['cast'].str.split(',')

plt.subplots(figsize=(12,10))

list1 = []

for i in movies['cast']:

    list1.extend(i)

ax = pd.Series(list1).value\_counts()[:15].sort\_values(ascending=True).plot.barh(width=0.9, color=sns.color\_palette('muted', 40))

for i, v in enumerate(pd.Series(list1).value\_counts()[:15].sort\_values(ascending=True).values):

    ax.text(.8, i, v, fontsize=10, color='white', weight='bold')

plt.title('Actors with highest appearance')

plt.show()

for i, j in zip(movies['cast'], movies.index):

    list2 = []

    list2 = i[:4]

    movies.loc[j, 'cast'] = str(list2)

movies['cast'] = movies['cast'].str.strip('[]').str.replace(' ', '').str.replace("'", '')

movies['cast'] = movies['cast'].str.split(',')

for i, j in zip(movies['cast'], movies.index):

    list2 = []

    list2 = i

    list2.sort()

    movies.loc[j, 'cast'] = str(list2)

movies['cast'] = movies['cast'].str.strip('[]').str.replace(' ', '').str.replace("'", '')

castList = []

for index, row in movies.iterrows():

    cast = row["cast"]

    for i in cast:

        if i not in castList:

            castList.append(i)

def binary(cast\_list):

    binaryList = []

    for genre in castList:

        if genre in cast\_list:

            binaryList.append(1)

        else:

            binaryList.append(0)

    return binaryList

movies['cast\_bin'] = movies['cast'].apply(lambda x: binary(x))

def xstr(s):

    if s is None:

        return ''

    return str(s)

movies['director'] = movies['director'].apply(xstr)

plt.subplots(figsize=(12,10))

ax = movies[movies['director'] != ''].director.value\_counts()[:10].sort\_values(ascending=True).plot.barh(width=0.9, color=sns.color\_palette('muted', 40))

for i, v in enumerate(movies[movies['director'] != ''].director.value\_counts()[:10].sort\_values(ascending=True).values):

    ax.text(.5, i, v, fontsize=12, color='white', weight='bold')

plt.title('Directors with highest movies')

plt.show()

directorList = []

for i in movies['director']:

    if (i is not None) and (i not in directorList):

        directorList.append(i)

def binary(director\_list):

    binaryList = []

    for direct in directorList:

        if direct in director\_list:

            binaryList.append(1)

        else:

            binaryList.append(0)

    return binaryList

movies['director\_bin'] = movies['director'].apply(lambda x: binary(x))

movies['keywords'] = movies['keywords'].str.strip('[]').str.replace(' ', '').str.replace("'", '').str.replace('"', '')

movies['keywords'] = movies['keywords'].str.split(',')

for i, j in zip(movies['keywords'], movies.index):

    list2 = []

    list2 = i

    movies.loc[j, 'keywords'] = str(list2)

movies['keywords'] = movies['keywords'].str.strip('[]').str.replace(' ', '').str.replace("'", '')

movies['keywords'] = movies['keywords'].str.split(',')

for i, j in zip(movies['keywords'], movies.index):

    list2 = []

    list2 = i

    list2.sort()

    movies.loc[j, 'keywords'] = str(list2)

movies['keywords'] = movies['keywords'].str.strip('[]').str.replace(' ', '').str.replace("'", '')

movies['keywords'] = movies['keywords'].str.split(',')

words\_list = []

for index, row in movies.iterrows():

    genres = row["keywords"]

    for genre in genres:

        if genre not in words\_list:

            words\_list.append(genre)

def binary(words):

    binaryList = []

    for genre in words\_list:

        if genre in words:

            binaryList.append(1)

        else:

            binaryList.append(0)

    return binaryList

movies['words\_bin'] = movies['keywords'].apply(lambda x: binary(x))

movies = movies[(movies['vote\_average'] != 0)]

movies = movies[movies['director'] != '']

def Similarity(movieId1, movieId2):

    a = movies.iloc[movieId1]

    b = movies.iloc[movieId2]

    genresA = a['genres\_bin']

    genresB = b['genres\_bin']

    genreDistance = spatial.distance.cosine(genresA, genresB)

    scoreA = a['cast\_bin']

    scoreB = b['cast\_bin']

    scoreDistance = spatial.distance.cosine(scoreA, scoreB)

    directA = a['director\_bin']

    directB = b['director\_bin']

    directDistance = spatial.distance.cosine(directA, directB)

    wordsA = a['words\_bin']

    wordsB = b['words\_bin']

    wordsDistance = spatial.distance.cosine(wordsA, wordsB)

    return genreDistance + directDistance + scoreDistance + wordsDistance

print(Similarity(3, 160))

print(movies.iloc[3])

print(movies.iloc[160])

new\_id = list(range(0, movies.shape[0]))

movies['new\_id'] = new\_id

movies = movies[['original\_title', 'genres', 'vote\_average', 'genres\_bin', 'cast\_bin', 'new\_id', 'director', 'director\_bin', 'words\_bin']]

import operator

def predict\_score(name):

    new\_movie = movies[movies['original\_title'].str.contains(name)].iloc[0].to\_frame().T

    print('Selected Movie: ', new\_movie.original\_title.values[0])

    def getNeighbors(baseMovie, K):

        distances = []

        for index, movie in movies.iterrows():

            if movie['new\_id'] != baseMovie['new\_id'].values[0]:

                dist = Similarity(baseMovie['new\_id'].values[0], movie['new\_id'])

                distances.append((movie['new\_id'], dist))

        distances.sort(key=operator.itemgetter(1))

        neighbors = []

        for x in range(K):

            neighbors.append(distances[x])

        return neighbors

    K = 10

    avgRating = 0

    neighbors = getNeighbors(new\_movie, K)

    print('\nRecommended Movies: \n')

    for neighbor in neighbors:

        avgRating = avgRating + movies.iloc[neighbor[0]][2]

        print(movies.iloc[neighbor[0]][0] + " | Genres: " + str(movies.iloc[neighbor[0]][1]).strip('[]').replace(' ', '') + " | Rating: " + str(movies.iloc[neighbor[0]][2]))

    print('\n')

    avgRating = avgRating / K

    print('The predicted rating for %s is: %f' % (new\_movie['original\_title'].values[0], avgRating))

    print('The actual rating for %s is %f' % (new\_movie['original\_title'].values[0], new\_movie['vote\_average']))

predict\_score('Godfather')

predict\_score('Donnie Darko')

predict\_score('Notting Hill')

A graph with different colored bars

Description automatically generated with medium confidence

A graph with colorful bars

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated